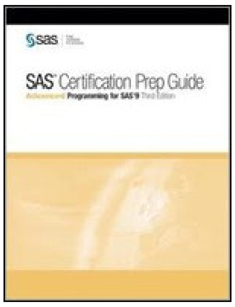


SAS Certification Prep Guide: Advanced Programming for SAS 9, Third Edition

by SAS Institute

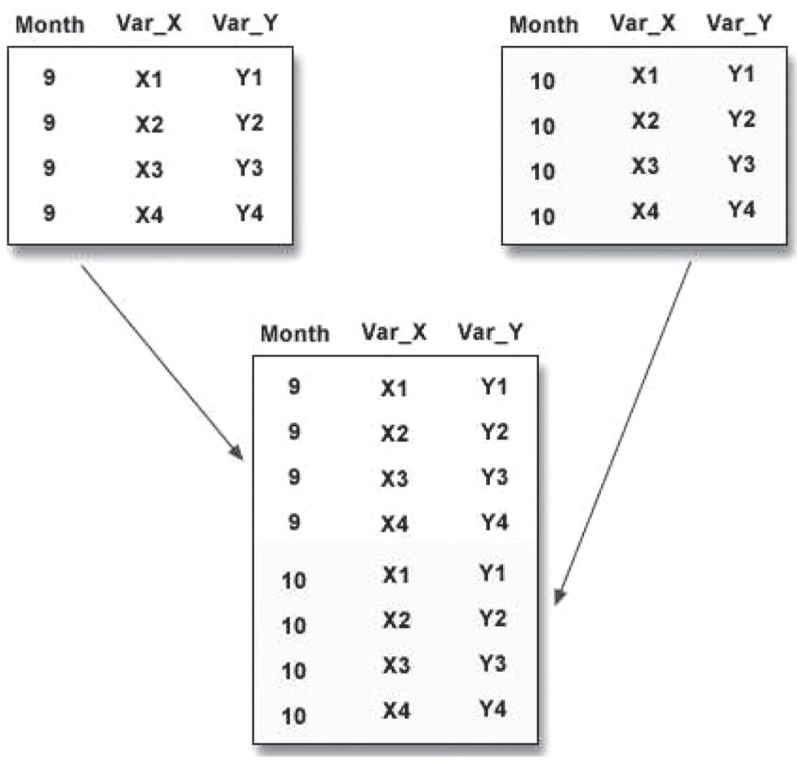
Chapter 14: Combining Data Vertically

Overview

Introduction

Combining data vertically refers to the process of concatenating or interleaving data. In some cases the data might be in SAS data sets. In other cases the data might be stored in raw data files.

In this chapter you learn how to create a SAS data set by concatenating multiple raw data files using the FILENAME and INFILE statements. You also learn how to concatenate SAS data sets using PROC APPEND.



Objectives

In this chapter you learn to

- create a SAS data set from multiple raw data files using a FILENAME statement
- create a SAS data set from multiple raw data files using an INFILE statement with the FILEVAR= option
- append SAS data sets using the APPEND procedure.

Using a FILENAME Statement

Overview

You already know that you can use a FILENAME statement to associate a fileref with a single raw data file. You can also use a FILENAME statement to concatenate raw data files by assigning a single fileref to the raw data files that you want to combine.

General form, FILENAME statement:

FILENAME *fileref ('external-file1' 'external-file2' …'external-filen');*

where

fileref

is any SAS name that is eight characters or fewer.

external-file

is the physical name of an external file. The physical name is the name that is recognized by the operating environment.

Caution All of the file specifications must be enclosed in one set of parentheses.

When the fileref is specified in an INFILE statement, each raw data file that has been referenced can be sequentially read into a data set using an INPUT statement.

Tip If you are not familiar with the content and structure of your raw data files, you can use PROC FSLIST to view them.

Example

In the following program, the FILENAME statement creates the fileref *Qtr1*, which references the raw data files *Month1.dat, Month2.dat, and Month3.dat*. The files are stored in the *C:\Sasuser* directory in the Windows operating environment. In the DATA step, the INFILE statement identifies the fileref, and the INPUT statement describes the data, just as if *Qtr1* referenced a single raw data file.

```
filename qtr1 ('c:\sasuser\month1.dat''c:\sasuser\month2.dat'
               'c:\sasuser\month3.dat');
data work.firstqtr;
   infile qtr1;
   input Flight $ Origin $ Dest $
         Date : date9. RevCargo : comma15.2;
run;
```

Table 14.1: RAW Data File Month1.dat (first five records)

```
----+----10---+----20---+----30---+----40
IA10200 SYD HKG 01JAN2000 $191,187.00
IA10201 SYD HKG 01JAN2000 $169,653.00
IA10300 SYD CBR 01JAN2000 $850.00
IA10301 SYD CBR 01JAN2000 $970.00
IA10302 SYD CBR 01JAN2000 $1,030.00
```

Table 14.2: Raw Data File Month2.dat (first five records)

```
----+----10---+----20---+----30---+----40
IA10200 SYD HKG 01MAR2000 $181,293.00
IA10201 SYD HKG 01MAR2000 $173,727.00
IA10300 SYD CBR 01MAR2000 $1,150.00
IA10301 SYD CBR 01MAR2000 $910.00
IA10302 SYD CBR 01MAR2000 $1,170.00
```

Table 14.3: Raw Data File Month3.dat (first five records)

```
----+----10---+----20---+----30---+----40
IA10200 SYD HKG 01FEB2000 $177,801.00
IA10201 SYD HKG 01FEB2000 $174,891.00
IA10300 SYD CBR 01FEB2000 $1,070.00
IA10301 SYD CBR 01FEB2000 $1,310.00
IA10302 SYD CBR 01FEB2000 $850.00
```

The SAS log indicates that the raw data files referenced by *Qtr1* are sequentially read into the SAS data set *Work.FirstQtr*.

Table 14.4: SAS Log

```
9 filename qtr1 ('c:\sasuser\month1.dat''c:\sasuser\month2.dat'
10                 'c:\sasuser\month3.dat');
11 data work.firstqtr;
12    infile qtr1;
13    input Flight $ Origin $ Dest $
14    Date : date9. RevCargo : comma15.2;
15 run;

NOTE: The infile QTR1 is:
      File Name=c:\sasuser\month1.dat,
      File List=('c:\sasuser\month1.dat' 'c:\sasuser\month2.dat'
                 'c:\sasuser\month3.dat'),
      RECFM=V,LRECL=256

NOTE: The infile QTR1 is:
      File Name=c:\sasuser\month2.dat,
      File List=('c:\sasuser\month1.dat' 'c:\sasuser\month2.dat'
                 'c:\sasuser\month3.dat'),
      RECFM=V,LRECL=256

NOTE: The infile QTR1 is:
      File Name=c:\sasuser\month3.dat,
      File List=('c:\sasuser\month1.dat' 'c:\sasuser\month2.dat'
                 'c:\sasuser\month3.dat'),
      RECFM=V,LRECL=256

NOTE: 50 records were read from the infile QTR1.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 50 records were read from the infile QTR1.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: 50 records were read from the infile QTR1.
      The minimum record length was 33.
      The maximum record length was 37.
NOTE: The data set WORK.FIRSTQTR has 150 observations
      and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           4.02 seconds
      cpu time            0.93 seconds
```

The following PROC PRINT output shows a portion of the observations in the *Work.FirstQtr* data set.

```
proc print
     data=work.firstqtr (firstobs=45 obs=55);
     format date date9.
            revcargo dollar11.2;
run;
```

Obs	Flight	Origin	Dest	Date	RevCargo
45	IA03505	RDU	BNA	01JAN2000	$2,697.00
46	IA03904	RDU	MCI	01JAN2000	$4,161.00
47	IA04503	LHR	GLA	01JAN2000	$3,498.00
48	IA04703	LHR	FRA	01JAN2000	$3,225.00
49	IA05002	BRU	LHR	01JAN2000	$1,989.00
50	IA05003	BRU	LHR	01JAN2000	$2,379.00
51	IA10200	SYD	HKG	01FEB2000	$177,801.00

52	IA10201	SYD	HKG	01FEB2000	$174,891.00
53	IA10300	SYD	CBR	01FEB2000	$1,070.00
54	IA10301	SYD	CBR	01FEB2000	$1,310.00
55	IA10302	SYD	CBR	01FEB2000	$850.00

Using an INFILE Statement

Overview

You can make the process of concatenating raw data files more flexible by using an INFILE statement with the FILEVAR= option. The FILEVAR= option enables you to dynamically change the currently opened input file to a new input file.

General form, INFILE statement with the FILEVAR= option:

INFILE *file-specification***FILEVAR=** *variable;*

where

FILEVAR= *variable*

names a variable whose change in value causes the INFILE statement to close the current input file and open a new input file.

variable

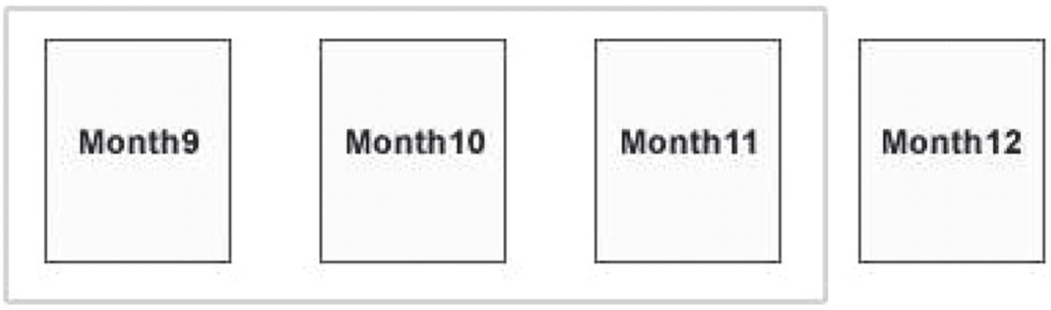
contains a character string that is a physical filename.

When you use an INFILE statement with the FILEVAR= option, the file specification is a placeholder, not an actual filename or a fileref that had been assigned previously to a file. SAS uses this placeholder for reporting processing information to the SAS log. The file specification must conform to the same rules as a fileref.

When the INFILE statement executes, it reads from the file that the FILEVAR= variable specifies. Like automatic variables, this variable is not written to the data set.

Example

Suppose you want to create a SAS data set that contains three months of data stored in three raw data files. The three months are the current month and the previous two months.



In the following INFILE statement, **`temp`** is an arbitrarily named placeholder, not an actual filename or a fileref that had been assigned to a file previously. The FILEVAR= variable **`nextfile`** contains the name of the raw data file to be read (for example, *Month9.dat, Month10.dat*, or *Month11.dat)*. A RUN statement is not included because the program is not complete.

```
data work.quarter;
   infile temp filevar=nextfile;
   input Flight $ Origin $ Dest $
         Date : date9. RevCargo : comma15.2;
```

Table 14.5: Raw Data File Month9.dat (first five records)

```
----+----10---+----20---+----30---+----40
IA10200 SYD HKG 01SEP2000 $189,441.00
IA10201 SYD HKG 01SEP2000 $175,473.00
IA10300 SYD CBR 01SEP2000 $1,370.00
IA10301 SYD CBR 01SEP2000 $710.00
IA10302 SYD CBR 01SEP2000 $1,210.00
```

Table 14.6: Raw Data File Month10.dat (first five records)

```
----+----10---+----20---+----30---+----40
IA10200 SYD HKG 01OCT2000 $182,457.00
IA10201 SYD HKG 01OCT2000 $160,923.00
IA10300 SYD CBR 01OCT2000 $1,030.00
IA10301 SYD CBR 01OCT2000 $870.00
IA10302 SYD CBR 01OCT2000 $770.00
```

Table 14.7: Raw Data File Month11.dat (first five records)

```
----+----10---+----20---+----30---+----40
IA10200 SYD HKG 01NOV2000 $176,637.00
IA10201 SYD HKG 01NOV2000 $164,997.00
IA10300 SYD CBR 01NOV2000 $1,230.00
IA10301 SYD CBR 01NOV2000 $1,230.00
IA10302 SYD CBR 01NOV2000 $790.00
```

Note You can also use multiple INFILE statements or operating system techniques to combine raw data files. However, this chapter discusses only the FILENAME statement and the INFILE statement with the FILEVAR= option.

Assigning the Names of the Files to Be Read

The next step is to assign the names of the three files to be read to the variable **`nextfile:`**

```
data work.quarter;
   infile temp filevar=nextfile;
   input Flight $ Origin $ Dest $
         Date : date9. RevCargo : comma15.2;
```

In this ease, use the raw data files *Month9.dat, Month10.dat*, and *Month11.dat*. Notice that the titles of the raw data files are very similar. They each start with *"Month"* and are followed by numeric characters and the file extension *.dat:*

- Month9.dat
- Month10.dat
- Month11.dat

You can use an iterative DO loop and the PUT function to automatically change the values assigned to **`nextfile.`**

Example

In the following code, the DO statement creates the index variable **`i`** and assigns it the values of *9, 10*, and *11*. The assignment statement then assigns the name of the raw data file to **`nextfile`** using the current value of **`i`** and the PUT function.

Month9.dat, Month10.dat, and *Month11.dat* are stored in the *C:\Sasuser* directory in the Windows operating environment. On the right side of the assignment statement, the text string *c:\sasuser\month* is concatenated with the current value of **`i`** using the double exclamation point (!!) concatenation operator, *c:\sasuser\month* **`i`** is then concatenated with the text string *.dat*.

```
data work.quarter;
   do i = 9, 10, 11;
      nextfile="c:\sasuser\month"
                !!put(i,2.)!!".dat";
      infile temp filevar=nextfile;
      input Flight $ Origin $ Dest $
            Date : date9. RevCargo : comma15.2;
   end;
```

The following table shows the value of **`nextfile`** as the value of **`i`** changes.

When i=	nextfile=
9	*c:\sasuser\month 9.dat*
10	*c:\sasuser\month10.dat*
11	*c:\sasuser\month11.dat*

Tip Depending on the characters available on your keyboard, the symbol you use as the concatenation operator can be a double vertical bar (||), a double broken vertical bar (||), or a double exclamation point (!!).

Using the COMPRESS Function

Note the space between *month* and *9* in *c:\sasuser\month 9.dat*. You can eliminate the space by using the COMPRESS function.

When i=	nextfile=
9	*c:\sasuser\month 9.dat*
10	*c:\sasuser\month10.dat*
11	*c:\sasuser\month11.dat*

General form, COMPRESS function:

COMPRESS*(source, <characters-to-remove>);*

where

source

specifies a source string that contains the characters to remove.

characters-to-remove

specifies the character or characters that SAS removes from the source string.

Note If the *characters-to-remove* is omitted, the COMPRESS function removes blank spaces from the source.

Example

In the following code, the COMPRESS function removes blank spaces from the value of **`nextfile`:**

```
data work.quarter;
   do i = 9, 10, 11;
      nextfile ="c:\sasuser\month"Mput(i,2.)!!".dat";
      nextfile=compress (nextfile,' ');
      infile temp filevar=nextfile;
      input Flight $ Origin $ Dest $
            Date : date9. RevCargo : comma15.2;
      end;
```

The COMPRESS function can be combined with the assignment statement for greater efficiency:

```
data work.quarter;
   do i = 9, 10, 11;
      nextfile="c:\sasuser\month"!!compress(put(i,2.)!!".dat",' ');
      infile temp filevar=nextfile;
      input Flight $ Origin $ Dest $
            Date : date9. RevCargo : comma15.2;
      end;
```

With the addition of the COMPRESS function, when the value of **`i`** equals *9*, **`nextfile`** is assigned the correct value, *c:\sasuser\month9.dat*.

When i=	nextfile=
9	*c:\sasuser\month9.dat*
10	*c:\sasuser\month10.dat*
11	*c:\sasuser\month11.dat*

You can add more statements to the program. An OUTPUT statement within the DO loop outputs each record to the SAS data set *Work.Quarter* and a STOP statement prevents an infinite loop of the DATA step.

```
data work.quarter;
   do i = 9, 10, 11;
      nextfile="c:\sasuser\month"
               !!compress(put(i,2.)!!".dat",' ');
      infile temp filevar=nextfile;
      input Flight $ Origin $ Dest $ Date : date9.
            RevCargo : comma15.2;
      output;
   end;
   stop;
```

The program is almost complete. However, there are several other statements that need to be added in order to read all of the input data.

Using the END= Option

When you read past the last record in a raw data file, the DATA step normally stops processing. In this case, you need to read the last record in the first two raw data files. However, you do not want to read past the last record in either of those files because the DATA step will stop processing. You can use the END= option with the INFILE statement to determine when you are reading the last record in the last raw data file.

General form, INFILE statement with the END= option:

INFILE *file-specification* **END=***variable;*

where

variable

names a variable that SAS sets to

- *0* when the current input data record *is not* the last record in the input file
- *1* when the current input record *is* the last record in the input file.

Note Like automatic variables, the END= variable is not written to the SAS data set.

The END= option enables you to name a variable whose value is controlled by SAS. The value of the variable is *0* when you are *not* reading the last record in an input file and *1* when you *are* reading the last record in an input file. You can test the value of the END= variable to determine whether the DATA step should continue processing.

Example

The END= variable **`lastobs`** is created in the INFILE statement. The DO UNTIL statement conditionally executes until the value of **`lastobs`** equals *1*. A RUN statement completes the program.

```
data work.quarter;
   do i = 9, 10, 11;
   nextfile="c:\sasuser\month"
            !!compress(put(i,2.)M".dat",' ');
         do until (lastobs);
         infile temp filevar=nextfile end=lastobs;
         input Flight $ Origin $ Dest $ Date : date9.
               RevCargo : comma15.2;
         output;
      end;
   end;
   stop;
run;
```

PROC PRINT output shows a portion of the observations in the SAS data set *Work.Quarter*. A LABEL statement is used to assign the descriptive label *Month* to the variable **`i`**. Notice that the variables **`nextfile`** and **`lastobs`** are not written to the data set.

```
proc print
    data=work.quarter
   (firstobs=45 obs=55) label;
 label i='Month';
 format date date9.
        revcargo dollar11.2;
run;
```

Obs	Month	Flight	Origin	Dest	Date	RevCargo
45	9	IA04300	LHR	CDG	01SEP2000	$1,750.00
46	9	IA05002	BRU	LHR	01SEP2000	$1,859.00
47	9	IA05005	BRU	LHR	01SEP2000	$2,197.00
48	9	IA05101	LHR	GVA	01SEP2000	$3,741.00
49	9	IA05202	GVA	LHR	01SEP2000	$4,089.00
50	9	IA05204	GVA	LHR	01SEP2000	$3,741.00
51	10	IA10200	SYD	HKG	01OCT2000	$182,457.00
52	10	IA10201	SYD	HKG	01OCT2000	$160,923.00
53	10	IA10300	SYD	CBR	01OCT2000	$1,030.00
54	10	IA10301	SYD	CBR	01OCT2000	$870.00
55	10	IA10302	SYD	CBR	01OCT2000	$770.00

Using Date Functions

You can make your program more flexible by eliminating the need to include explicit month numbers in your SAS statements. To create a program that will always read the *current* month and the previous two months, you can use date functions to obtain the month number of today's date to begin the quarter.

Example

In the following program, the MONTH and TODAY functions are used to obtain the value of the variable **`monthnum`**. The TODAY function returns the current date from the system clock as a SAS date value. The month number is then extracted from the current date using the MONTH function.

The value of **`midmon`** is calculated by subtracting 1 from the value of **`monthnum`**. The value of **`lastmon`** is then calculated by subtracting 1 from the values of **`midmon`**. The following table shows the values **`monthnum, midmon`**, and **`lastmon`** if the current date is October 22, 2002.

Variable	Value
monthnum	*10*
midmon	*9*
lastmon	*8*

In the previous example, the DO statement created the index variable **i** and assigned it the values of *9*,*10*, and *11*. Here, the DO statement assigns **i** the values of **monthnum, midmon**, and **lastmon:**

```
data work.quarter (drop=monthnum midmon lastmon);
   monthnum=month(today());
   midmon=monthnum-1;
   lastmon=midmon-1;
   do i = monthnum, midmon, lastmon;
      nextfile="c:\sasuser\month"
               !!compress(put(i,2.)!!".dat",' ');
      do until (lastobs);
         infile temp filevar=nextfile end=lastobs;
         input Flight $ Origin $ Dest $ Date : date9.
               RevCargo : comma15.2;
         output;
      end;
   end;
   stop;
run;
```

The following PROC PRINT output shows a portion of the observations in *Work.Quarter*.

```
proc print data=work.quarter
           (firstobs=45 obs=55) label;
   label i='Month';
   format date date9.
          revcargo dollar11.2;
run;
```

Obs	Month	Flight	Origin	Dest	Date	RevCargo
45	4	IA02705	RDU	MIA	01APR2000	$3,828.00
46	4	IA02800	MIA	RDU	01APR2000	$5,148.00
47	4	IA02804	MIA	RDU	01APR2000	$5,852.00
48	4	IA02805	MIA	RDU	01APR2000	$4,180.00
49	4	IA03200	ANC	SFO	01APR2000	$45,974.00
50	4	IA03401	ANC	RDU	01APR2000	$95,914.00
51	3	IA10200	SYD	HKG	01MAR2000	$181,293.00
52	3	IA10201	SYD	HKG	01MAR2000	$173,727.00
53	3	IA10300	SYD	GBR	01MAR2000	$1,150.00
54	3	IA10301	SYD	GBR	01MAR2000	$910.00
55	3	IA10302	SYD	GBR	01MAR2000	$1,170.00

Using the INTNX Function

In the previous example the current month was October. What happens if the current month is January or February?

Suppose the current date is February 16, 2003. Using the following program, the values for **midmon** (January) and **lastmon** (December) would be *1* and *0* respectively. Since there is no "0" month, the program would fail to read the third raw data file.

```
data work.quarter (drop=monthnum midmon lastmon);
   monthnum=month(today());
   midmon=monthnum-1;
   lastmon=midmon-1;
   do i = monthnum, midmon, lastmon;
      nextfile="c:\sasuser\month"
                !!compress(put(i,2.)!!".dat",' ');
      do until (lastobs);
         infile temp filevar=nextfile end=lastobs;
         input Flight $ Origin $ Dest $ Date : date9
               RevCargo : comma15.2;
         output;
      end;
   end;
   stop;
run;
```

Variable	Value
monthnum	*2*
midmon	*1*
lastmon	*0*

You can use the INTNX function with the TODAY and MONTH functions to correctly determine the values of **`midmon`** and **`lastmon`** for any given date. Remember that the INTNX function increments a date, time, or datetime value by a given interval or intervals, and returns a date, time, or datetime value.

Example

Suppose the current date is January 30, 2003. In the following program **`monthnum`** is assigned a value of *1* using the TODAY and MONTH functions. The INTNX function is used with the TODAY and MONTH functions to assign a value of *12* to **`midmon`** and a value of *11* to **`lastmon`**.

```
data work.quarter (drop=monthnum midmon lastmon);
   monthnum=month(today());
   midmon=month(intnx('month',today(),-1));
   lastmon=month(intnx('month',today(),-2));
   do i = monthnum, midmon, lastmon;
      nextfile="c:\sas\month"!!compress(put(i,2.)!!".dat",' ')
      do until (lastobs);
         infile temp filevar=nextfile end=lastobs;
         input Flight $ Origin $ Dest $ Date : date9.
               RevCargo : comma15.2;
         output;
      end;
   end;
   stop;
run;
```

Variable	Value
monthnum	*1*
midmon	*12*
lastmon	*11*

The following PROC PRINT output shows a portion of the observations in *Work.Quarter*.

```
proc print data=work.quarter
           (firstobs=45 obs=55) label;
```

```
      label i='Month';
      format date date9.
             revcargo dollar11.2;
   run;
```

Obs	Month	Flight	Origin	Dest	Date	RevCargo
45	4	IA02705	RDU	MIA	01APR2000	$3,828.00
46	4	IA02800	MIA	RDU	01APR2000	$5,148.00
47	4	IA02804	MIA	RDU	01APR2000	$5,852.00
46	4	IA82805	MIA	RDU	01APR2000	$4,180.00
49	4	IA03200	ANC	SFO	01APR2000	$45,974.00
50	4	IA03401	ANC	RDU	01APR2000	$95,914.00
51	3	IA10200	SYD	HKG	01MAR2000	$181,293.00
52	3	IA10201	SYD	HKG	01MAR2000	$173,727.00
53	3	IA10300	SYD	CBR	01MAR2000	$1,150.00
54	3	IA10301	SYD	CBR	01MAR2000	$910.00
55	3	IA10302	SYD	CBR	01MAR2000	$1,170.00

Appending SAS Data Sets

Overview

Now that you have seen several methods for concatenating raw data files, we can consider how you can use the APPEND procedure to concatenate two SAS data sets.

General form, PROC APPEND:

PROC APPEND BASE=*SAS-data-set* **DATA**=*SAS-data-set;*

RUN;

where

BASE=*SAS-data-set*

names the data set to which you want to add observations.

DATA=SAS-data-set

names the SAS data set containing observations that you want to append to the end of the SAS data set specified in the BASE= argument.

PROC APPEND reads only the data in the DATA= SAS data set, not the BASE= SAS data set. PROC APPEND concatenates data sets even though there might be variables in the BASE= data set that do not exist in the DATA= data set.

When the BASE= data set contains more variables than the DATA= data set, missing values for the additional variables are assigned to the observations that are read in from the DATA= data set and a warning message is written to the SAS log.

Example

The SAS data sets *Work.Cap2001* and *Work.Capacity* both contain the following variables: **`Caplst, CapBusiness. CapEcon, Dest, FlightID, Origin`**, and **`RouteID`**. However, the BASE= data set *(Work.Cap2001)* contains an additional variable, **`Date`**, that is not included in the DATA= data set *(Work.Capacity)*.

When the following program is submitted, the SAS log indicates that the variable **`Date`** was not found in the DATA= file.

```
proc append base=work.cap2001
            data=work.capacity;
run;
```

Table 14.8: SAS Log

```
NOTE: Appending WORK.CAPACITY to WORK.CAP2001.
WARNING: Variable Date was not found on DATA file.
NOTE: There were 50 observations read from the data set WORK.CAPACITY.
NOTE: 50 observations added.
NOTE: The data set WORK.CAP2001 has 100 observations and 8 variables.
NOTE: PROCEDURE APPEND used (Total process time):
```

PROC PRINT output of the appended version of *Work.Cap2001* shows that missing values have been assigned to **`Date`** in the observations that were read in from the DATA= data set.

```
proc print data=work.cap2001
           (firstobs=45 obs=55);
run
```

Obs	FlightID	RouteID	Origin	Dest	Cap 1st	CapBusiness	CapEcon	Date
45	IA02503	0000025	RDU	IND	12	0	138	21JAN2001
46	IA02504	0000025	RDU	IND	12	0	138	24JAN2001
47	IA02600	0000026	IND	RDU	12	0	138	02JAN2001
48	IA02605	0000026	IND	RDU	12	0	138	05JAN2001
49	IA02603	0000026	IND	RDU	12	0	138	16JAN2001
50	IA02703	0000027	RDU	MIA	12	0	138	17JAN2001
51	IA00100	0000001	RDU	LHR	14	30	163	.
52	IA00201	0000002	LHR	RDU	14	30	163	.
53	IA00300	0000003	RDU	FRA	14	30	163	.
54	IA00400	0000004	FRA	RDU	14	30	163	.
55	IA00500	0000005	RDU	JFK	16	.	251	.

Note You can also use the DATA step SET statement to combine SAS data vertically. If multiple data set names appear in the SET statement, the resulting output data set is a concatenation of all the data sets listed. Unlike the APPEND procedure, the SET statement in the DATA step reads all observations in both input data sets in order to concatenate them. Therefore, the APPEND procedure is more efficient than the SET statement in the DATA step for concatenating data sets because it reads only the data in the DATA= data set.

In the following program, SAS reads all of the observations from *Work.Cap2001*, then all of the observations from *Work.Capacity*.

```
data work.new;
   set work.cap2001 work.capacity;
run;
```

Note You can also use the SQL procedure to combine SAS data vertically. For information about using the SQL procedure to combine data vertically, see "Combining Tables Vertically Using PROC SQL" on page 132.

Using the FORCE Option

In the previous example, the DATA= data set *(Work.Capacity)* contained *fewer* variables than the BASE= data set *(Work.Cap2001)*. However, you might need to append data sets when the DATA= data set contains *more* variables than the BASE= data set.

You must use the FORCE option with the APPEND procedure to concatenate data sets when the DATA= data set contains variables that are not in the BASE= data set.

General form, PROC APPEND with the FORCE option:

PROC APPEND BASE=*SAS-data-set* **DATA**=*SAS-data-set* <FORCE>;

Caution The FORCE option can cause loss of data due to truncation or dropping of variables.

When you use the FORCE option, the structure of the BASE=data set is used for the appended data set.

Example

Remember that the SAS data sets *Work.Cap2001* and *Work.Capacity* both contain the followingvariables: **Cap1st, CapBusiness, CapEcon, Dest, FlightID, Origin**, and **RouteID**. In this case, the *DATA*= data set *(Work.Cap2001)* contains an additional variable, **Date**, that is not included in the *BASE*= data set *(Work.Capacity)*.

When the following program is submitted, the SAS log indicates that the data sets were not appended because the variable **Date** was not found in the BASE= file.

```
proc append base=work.capacity
            data=work.cap20 01;
run;
```

Table 14.9: SAS Log

```
NOTE: Appending WORK.CAP2001 to WORK.CAPACITY.
WARNING: Variable Date was not found on BASE file.
ERROR: No appending done because of anomalies listed above.
        Use FORCE option to append these files.
NOTE: 0 observations added.
NOTE: The data set WORK.CAPACITY has 50 observations and 7 variables.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.02 seconds
      cpu time            0.03 seconds
NOTE: The SAS System stopped processing this step because of errors.
```

When the FORCE option is used with PROC APPEND, the SAS log indicates that observations have been read from the DATA= data set, but that dropping or truncating will occur.

```
proc append base=work.capacity
            data=work.cap20 01 force;
run;
```

Table 14.10: SAS Log

```
NOTE: Appending WORK.CAP2001 to WORK.CAPACITY.
WARNING: Variable Date was not found on BASE file.
NOTE: FORCE is specified, so dropping/truncating will occur.
NOTE: There were 50 observations read from the data set WORK.CAP2001.
NOTE: 50 observations added.
NOTE: The data set WORK.CAPACITY has 100 observations and 7 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds
```

PROC PRINT output shows that the variable **Date** has been dropped from the appended version of *Work.Capacity*.

```
proc print
     data=work.capacity
         (firstobs=45 obs=55)
```

```
run;
```

Obs	FlightID	RouteID	Origin	Dest	Cap1st	CapBusiness	CapEcon
45	IA04500	0000045	LHR	GLA	14	.	125
46	IA04600	0000046	GLA	LHR	14	.	125
47	IA04700	0000047	LHR	FRA	14	.	125
48	IA04800	0000048	FRA	LHR	14	.	125
49	IA04900	0000049	LHR	BRU	14	.	125
50	IA05000	0000050	BRU	LHR	14	.	125
51	IA00100	0000001	RDU	LHR	14	30	163
52	IA00101	0000001	RDU	LHR	14	30	163
53	IA00201	0000002	LHR	RDU	14	30	163
54	IA00301	0000003	RDU	FRA	14	30	163
55	IA00603	0000006	JFK	RDU	16	34	251

Appending Variables with Different Lengths

If the DATA= data set contains variables that are longer than the variables in the BASE= data set, the FORCE option must be used with PROC APPEND. Using the FORCE option enables you to append the data sets. However, the DATA= variable values will be truncated.

Example

In the SAS data set *Work.Acities*, the variable **`City`** has a length of 22. In the SAS data set *Work.WestAust*, **`City`** has a length of 50. You can use the CONTENTS procedure to view the attributes of the variables in each data set.

```
proc contents data=work.acities;
run;
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
1	City	Char	22	City Where Airport is Located
2	Code	Char	3	Start Point
4	Country	Char	40	Country Where Airport is Located
3	Name	Char	50	Airport Name

```
proc contents data=work.westaust;
run;
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
2	City	Char	50	City Where Airport is Located
1	Code	Char	3	Airport Code
3	Country	Char	40	Country Where Airport is Located
4	Name	Char	50	Airport Name

When the following program is submitted, the SAS log indicates that the data sets were not appended due to different lengths for **City** in the BASE= and DATA= data sets.

```
proc append base=work.acities
            data=work.westaust;
run;
```

Table 14.11: SAS Log

```
NOTE: Appending WORK.WESTAUST to WORK.ACITIES.
WARNING: Variable City has different lengths on BASE and
         DATA files (BASE 22 DATA 50).
ERROR: No appending done because of anomalies listed above.
       Use FORCE option to append these files.
NOTE: 0 observations added.
NOTE: The data set WORK.ACITIES has 50 observations and 4 variables.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           1.44 seconds
      cpu time            0.06 seconds
NOTE: The SAS System stopped processing this step because of errors.
```

When the FORCE option is used, the SAS log indicates that the data sets are appended, but that dropping or truncating will occur.

```
proc append base=work.acities
            data=work.westaust force;
run;
```

Table 14.12: SAS Log

```
NOTE: Appending WORK.WESTAUST to WORK.ACITIES.
WARNING: Variable City has different lengths on BASE and DATA files
         (BASE 22 DATA 50).
NOTE: FORCE is specified, so dropping/truncating will occur.
NOTE: There were 50 observations read from the data set WORK.WESTAUST.
NOTE: 50 observations added.
NOTE: The data set WORK.ACITIES has 100 observations and 4 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           1.44 seconds
      cpu time            0.06 seconds
```

PROC CONTENTS output for the appended version of *Work.Acities* shows that the variable **City** has retained a length of 22 from the BASE= data set. Also notice that the variable **Code** has retained the label *Start Point* from the BASE= data set.

```
proc contents
     data=work.aciti
run;
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
1	City	Char	22	City Where Airport is Located
2	Code	Char	3	Start Point
4	Country	Char	40	Country Where Airport is Located
3	Name	Char	50	Airport Name

PROC PRINT output shows that some of the values of **City** are truncated in the appended version of *Work.Acities*.

```
proc print
     data=work.acities
          (firstobs=45 obs=55);
```

```
run;
```

Obs	City	Code	Name	Country
45	Portland, ME	PWM	Portland International Jetport	USA
46	Raleigh-Durham, NC	RDU	Raleigh-Durham International Airport	USA
47	Seattle, WA	SEA	Seattle-Tacoma International Airport	USA
48	San Francisco, CA	SFO	San Francisco International Airport	USA
49	Singapore	SIN	Changi International Airport	Singapore
50	Sydney, New South Wale	SYD	Kingsford Smith	Australia
51	Argyle Downs, Western	AGY		Australia
52	Albany, Western Austra	ALH		Australia
53	Big Bell, Western Aust	BBE		Australia
54	Bedford Downs, Western	BDW		Australia
55	Beagle Bay, Western Au	BEE		Australia

Appending Variables with Different Types

If the DATA= data set contains a variable that does not have the same type as the corresponding variable in the BASE= data set, the FORCE option must be used with PROC APPEND. Using the FORCE option enables you to append the data sets. However, missing values are assigned to the DATA= variable values for the variable whose type did not match.

Example

In the SAS data set *Work.Allemps*, the variable **`Phone`** is a character variable. In the SAS data set *Work.Newemps*, **`Phone`** is a numeric variable. You can use PROC CONTENTS to view the attributes of the variables in each data set.

```
proc contents data=work.allemps;
run;
```

Alplabetic List of Variables and Attributes			
#	Variable	Type	Len
5	Division	Char	30
1	EmpID	Char	6
2	LastName	Char	15
4	Location	Char	13
3	Phone	Num	8

```
proc contents data=work.newemps;
run;
```

Alplabetic List of Variables and Attributes			
#	Variable	Type	Len
5	Division	Char	30
1	EinpID	Char	6
2	LastName	Char	15
4	Location	Char	13
3	Phone	Char	4

When the following program is submitted, the SAS log indicates that there is a type mismatch for the variable **Phone** and that data sets were not appended.

```
proc append base=work.allemps
            data=work.newemps;
run;
```

Table 14.13: SAS Log

```
NOTE: Appending WORK.NEWEMPS to WORK.ALLEMPS.
WARNING: Variable Phone not appended because of type mismatch.
ERROR: No appending done because of anomalies listed above.
       Use FORCE option to append these files.
NOTE: 0 observations added.
NOTE: The data set WORK.ALLEMPS has 550 observations and 5 variables.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.08 seconds
      cpu time            0.01 seconds
NOTE: The SAS System stopped processing this step because of errors.
```

When the FORCE option is used, the SAS log indicates that the data sets are appended, but that the variable **`Phone`** is not appended due to the type mismatch.

```
proc append base=work.allemps
            data=work.newemps force;
run;
```

Table 14.14: SAS Log

```
NOTE: Appending WORK.NEWEMPS to WORK.ALLEMPS.
WARNING: Variable Phone not appended because of type mismatch.
NOTE: FORCE is specified, so dropping/truncating will occur.
NOTE: There were 19 observations read from the data set WORK.NEWEMPS.
NOTE: 19 observations added.
NOTE: The data set WORK.ALLEMPS has 569 observations and 5 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time          0.05 seconds
      cpu time           0.02 seconds
```

PROC CONTENTS output for the appended version of *Work.Allemps* shows that the variable **`Phone`** has retained the type of character from the BASE= data set.

```
proc contents
     data=work.allemps;
run;
```

Alpltabetic List of Varables and Attributes

#	Variable	Type	Len
5	Division	Char	30
1	EmpID	Char	6
2	LastName	Char	15
4	Location	Char	13
3	Phone	Char	4

PROC PRINT output of the appended version of *Work.Allemps* shows that the the values for **`Phone`** are missing in the records that were read in from the DATA= data set.

```
proc print
     data=work.allemps
    (firstobs=45 obs=55)
run;
```

Obs	EmpID	LastName	Phone	Location	Division
45	E00213	DICKEY	1519	CARY	AIRPORT OPERATIONS
46	E00226	BAUCOM	1124	CARY	AIRPORT OPERATIONS
47	E00231	SPENCER	2868	CARY	AIRPORT OPERATIONS
48	E00236	BAILEY	1088	CARY	AIRPORT OPERATIONS
49	E00243	FILIPOWSKI	1635	CARY	AIRPORT OPERATIONS
50	E00249	YUAN	3241	CARY	AIRPORT OPERATIONS
51	E00490	CANCELLO		ROME	FINANCE & IT
52	E00496	PRESTON		LONDON	FINANCE & IT
53	E00499	ZILSTORFF		COPENHAGEN	AIRPORT OPERATIONS
54	E00500	LEY		FRANKFURT	FINANCE & IT
55	E00503	BRAMMER		COPENHAGEN	SALES & MARKETING

Additional Features

In addition to the methods for appending raw data files that were discussed earlier in this chapter, you can also append raw data files using a SAS data set or an external file that contains the names of the raw data files to be appended.

Storing Raw Data Filenames in a SAS Data Set

In the following program, five raw data files, *Routel.dat, Route2.dat, Route3.dat, Route4.dat*, and *Route5.dat*, are concatenated to create the SAS data set *Work.NewRoute*. The names of the raw data files are stored in the SAS data set *Sasuser.Rawdata*, which is referenced using a SET statement. The name of the FILEVAR= variable, **`readit`**, is the name of the variable in *Sasuser.Rawdata* whose value is the name of the file to be read.

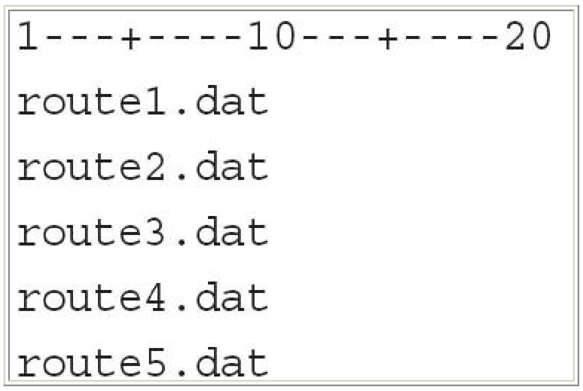
```
data work.newroute;
  set sasuser.rawdata;
  infile in filevar = readit end = lastfile;
  do while(lastfile = 0);
    input @1 RouteID $7. @8 Origin $3. @11 Dest $3
          @14 Distance 5. @19 Fare1st 4.
          @23 FareBusiness 4. @27 FareEcon 4.
          @31 FareCargo 5.;
    output;
  end;
run;
```

Obs	readit
1	route1.dat
2	route2.dat
3	route3.dat
4	route4.dat
5	route5.dat

Storing Raw Data Filenames in an External File

In the following program, *Routel.dat, Route2.dat, Route3.dat, Route4.dat*, and *Route5.dat* are also concatenated to create the SAS data set *Work.NewRoute*. In this example, the names of the raw data files are stored in the external file *Raw data files. dat*, which is referenced in the first INFILE statement. The name of the FILEVAR= variable, **`readit`**, is the name of the variable read from *Raw data files. dat*. The value of **`readit`** is the name of the raw data file to be read.

Table 14.15: Raw Data File Raw data files.dat



```
1---+----10---+----20
route1.dat
route2.dat
route3.dat
route4.dat
route5.dat
```

```
data work.newroute;
   infile 'Raw data files.dat';
   input readit $10.;
   infile in filevar=readit end=lastfile;
   do while(lastfile = 0);
     input @1 RouteID $7. @8 Origin $3. @11 Dest $3.
           @14 Distance 5. @19 Fare1st 4.
           @23 FareBusiness 4. @27 FareEcon 4.
           @31 FareCargo 5.;
     output;
   end;
run;
```

Summary

Contents

This section contains the following topics.

Text Summary

Using a FILENAME Statement

You can use a FILENAME statement to concatenate raw data files by assigning a single fileref to the raw data files that you want to combine. When the fileref is specified in an INFILE statement, each raw data file that has been referenced can be sequentially read into a data set using an INPUT statement.

Using an INFILE Statement

You can make the process of concatenating raw data files more flexible by using an INFILE statement with the FILEVAR= option. The FILEVAR= option enables you to dynamically change the currently opened input file to a new input file. When the INFILE statement executes, it reads from the file that the FILEVAR= variable specifies.

In some cases, you might need to use the COMPRESS function to eliminate spaces in the filenames you generate.

When you read the last record in a raw data file, the DATA step normally stops processing. When you are concatenating raw data files, you do not want to read past the last record until you reach the end of the last input file. You can determine whether you are reading the last record in the last raw data file by using the END= option with the INFILE statement. You can then test the value of the END= variable to determine whether the DATA step should continue processing.

If you are working with date-related data, you might be able to make your program more flexible by eliminating the need to include explicit month numbers in your SAS statements. To create a program that will always read the current month and the previous two months, you can use the MONTH and TODAY functions to obtain the month number of today's date to begin the quarter. In some cases, you might need to use the INTNX function with the TODAY and MONTH functions to correctly determine the month numbers.

Appending SAS Data Sets

You can use PROC APPEND to concatenate two SAS data sets. PROC APPEND reads only the data in the DATA= SAS data set, not in the BASE= SAS data set. PROC APPEND concatenates data sets even though there might be variables in the BASE= data set that do not exist in the DATA= data set.

The FORCE option must be used if the DATA= data set contains variables that

- are not in the BASE= data set
- are longer than the variables in the BASE= data set
- do not have the same type as the variables in the BASE= data set.

The FORCE option can cause loss of data due to truncation or dropping of variables. The following table summarizes the consequences of using the FORCE option.

DATA= data set contains variables that…	FORCE required?	Consequences of using the FORCE option
are in the BASE=data set, but the BASE= data set has more variables	no	Missing values are assigned to the extra BASE= data set variables.
are not in the BASE= data set	yes	Extra DATA= data set variables are dropped.
are longer than the variables in the BASE= data set	yes	DATA= data set variable values are truncated.
do not have the same type as the variables in the BASE= data set	yes	Missing values are assigned to the DATA= data set variables with the data type mismatch.

Additional Features

You can also append raw data files using a SAS data set or an external file that contains the names of the raw data files to be appended.

Syntax

Combining Raw Data Files Using a FILENAME Statement

```
FILENAME fileref ('external-filel' 'external-file2' 'external-filen');
DATA SAS-data-set;
  INFILE file-specification;
  INPUT  variable <$><&|:> <informat>;
RUN;
```

Combining Raw Data Files Using an INFILE Statement

```
DATA SAS-data=set;
  DO index-variable=variablel, variable2, variablen;
     variable = "text-string" !!PUT(index-variable,fiormat)!! "text-string";
     variable = COMPRESS (variable,' ');
     DO UNTIL(variable);
       INFILE file-specification FILEVAR=variable END=variable;
       INPUT variable <$> <&|:> <infiormat>;
       OUTPUT;
     END;
  END;
  STOP;
RUN;
```

Combining SAS Data Sets Using PROC APPEND

```
PROC APPEND BASE=SAS-data-set DATA=SAS-data-set <FORCE>;
RUN;
```

Sample Programs

Combining Raw Data Files Using a FILENAME Statement

```
filename qtr1 ('c:\data\month1.dat''c:\data\month2.dat'
               'c:\data\month3.dat');
data work.firstqtr;
   infile qtr1;'
   input Flight $ Origin $ Dest $
         Date : date9. RevCargo : comma15.2;
run;
```

Combining Raw Data Files Using an INFILE Statement

```
data quarter (drop=monthnum midmon lastmon);
   monthnum=month(today());
   midmon=month(intnx('month',today(),-1));
   lastmon=month(intnx('month',today(),-2));
   do i = monthnum, midmon, lastmon;
      nextfile="c:\sasuser\month"
               !!compress(put(i,2.)!!".dat",' ');
      do until (lastobs);
         infile temp filevar=nextfile end=lastobs;
         input Flight $ Origin $ Dest $ Date : date9.
               RevCargo : comma15.2;
         output;
      end;
   end;
   stop;
run;
```

Combining SAS Data Sets Using PROC APPEND

```
proc append base=work.acities
            data=work.airports force;
run;
```

Points to Remember

- When you use an INFILE statement with the FILEVAR= option, the file specification is just a placeholder, not an actual filename or a fileref that has been previously assigned to a file.
- Like automatic variables, the FILEVAR=variable and the END= variable are not written to the data set.
- Using the FORCE option with PROC APPEND can cause loss of data due to truncation or dropping of variables.
- When you use the FORCE option, the structure of the BASE= data set is used for the appended data set.

Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. Which of the following statements associates the fileref *OnSale* with the raw data files *London.dat, Paris.dat*, ? and *Zurich.dat*? The files are stored in the *C:\Routes \New* directory in the Windows operating environment.

a.
```
a. filename onsale (c:\routes\new\london.dat,
     c:\routes\new\paris.dat,
     c:\routes\new\zurich.dat);
```

b.
```
b. filename onsale 'c:\routes\new\london.dat'
     'c:\routes\new\paris.dat'
     'c:\routes\new\zurich.dat';
```

c.
```
c. filename onsale ('c:\routes\new\london.dat'
        'c:\routes\new\paris.dat'
        'c:\routes\new\zurich.dat');
```

d.
```
d. filename onsale 'c:\routes\new\london.dat
        c:\routes\new\paris.dat
        c:\routes\new\zurich.dat';
```

2. Which of the following statements is true? ?

a. The FILEVAR= option can be used to dynamically change the currently opened input file to a new physical file.

b. The FILEVAR= variable is not written to the data set.

c. The FILEVAR= variable must contain a character string that is a physical filename.

d. all of the above

3. Given the following program, which table correctly shows the corresponding values of the variables **x** and **readfile?** ?

```
data work.revenue;
   do x = 8, 9, 10;
      readfile=compress("c:\data\month"
        !!put(x,2.)!!".dat",' ');
      do until (lastobs);
         infile temp filevar=readfile
           end=lastobs;
         input Date : date7. Location $
               Sales : dollar10.2;
         output;
      end;
    end;
    stop;
run;
```

a.

When x=	readfile=
8	*month8.dat*
9	*month9.dat*
10	*month10.dat*

b.

When x=	readfile=
8	*c:\data\month8.dat*
9	*c:\data\month9.dat*
10	*c:\data\month10.dat*

c.

When x=	readfile=
8	*c:\data\month 8.dat*
9	*c:\data\month 9.dat*
10	*c:\data\month10.dat*

d.

When x=	readfile=
8	*month8*
9	*month9*
10	*month10*

4. If the current date is March 30, 2003, which table correctly shows the corresponding values of the variables ?

yl, y2, y3, and **nextfile?**

```
data work.quarter (drop=monthnum midmon lastmon);
   y3=year(today());
   y2=y3-1;
   y1=y3-2;
   do i = y3, y2, y1;
      nextfile="c:\data\Y"!!put(i,4.)!!".dat";
      do until (lastobs);
         infile temp filevar=nextfile
           end=lastobs;
         input Flight $ Origin $ Dest $
               Date : date9.;
         output;
      end;
   end;
   stop;
run;
```

a.

When i=	nextfile=
y1	*c:\data\Y2001.dat*
y2	*c:\data\Y2002.dat*
y3	*c:\data\Y2003.dat*

b.

When i=	nextfile=
y1	*Y2001.dat*
y2	*Y2002.dat*
y3	*Y2003.dat*

c.

When i=	nextfile=
y1	*c:\data\Y2003.dat*
y2	*c:\data\Y2002.dat*
y3	*c:\data\Y2001.dat*

d.

When i=	nextfile=
y1	*c:\data\Y3.dat*
y2	*c:\data\Y2.dat*
y3	*c:\data\Y1.dat*

5. What happens when SAS processes the last data record in an input file? ?

a. The END= variable is set to *1*.

b. The END= variable is set to *0*.

c. The END= variable is set to the number of records in the input file.

d. The END= variable is written to the SAS data set.

6. Which program appends *Work.London* to *Work.Flights?* ?

Data Set Description for Work.London

Variable	Type	Length
FlightNum	char	4
Destination	char	3
Departure	num	8
Gate	char	2

Data Set Description for Work.Flights

Variable	Type	Length
FlightNum	char	4
Destination	char	3
Departure	num	8
Gate	char	2

a.
```
a. proc append base=work.london
               data=work.flights;
   run;
```

b.
```
b. proc append data=work.london
               base=work.flights;
   run;
```

c.
```
c. proc append data=work.london work.flights;
   run;
```

d.
```
d. proc append data=work.flights work.london;
   run;
```

7. What happens when the following program is submitted? ?

```
proc append base=staff.marketing
            data=staff.sales force;
run;
```

Data Set Description for Staff.Marketing

Variable	Type	Length
LastName	char	12
FirstName	char	10
EmpID	char	5
Office	char	4
Phone	char	12

Data Set Description for Staff.Sales

Variable	Type	Length
LastName	char	20
FirstName	char	10
EmpID	char	5
Office	char	4
Phone	char	12

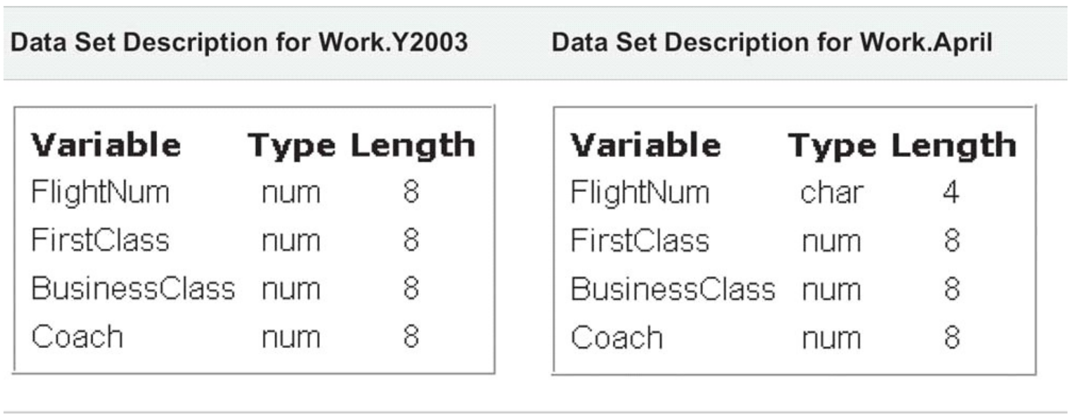
a. The length of **`LastName`** is converted to 20 in *Staff.Marketing*.

b. **`LastName`** is dropped from *Staff.Marketing*.

c. Missing values are assigned to **`LastName`** observations that are read in from *Staff.Sales*.

d. Some of the values of **`LastName`** might be truncated in the observations that are read in from *Staff.Sales*.

8. Which program appends *Work.April* to *Work.Y2003?* ?



Data Set Description for Work.Y2003

Variable	Type	Length
FlightNum	num	8
FirstClass	num	8
BusinessClass	num	8
Coach	num	8

Data Set Description for Work.April

Variable	Type	Length
FlightNum	char	4
FirstClass	num	8
BusinessClass	num	8
Coach	num	8

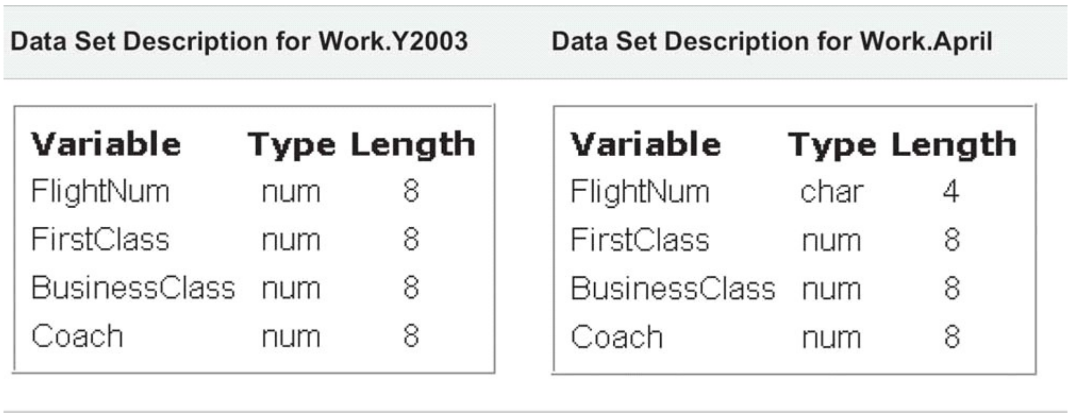
a.
```
a. proc append base=work.y2003
               data=work.april;
   run;
```

b.
```
b. proc append base=work.april
               data=work.y2003 force;
   run;
```

c.
```
c. proc append base=work.y2003
               data=work.april force;
   run;
```

d.
```
d. proc append base=work.april
               data=work.y2003;
   run;
```

9. What happens when the SAS data set *Work.NewHires* is appended to the SAS data set *Work.Employees* using PROC APPEND? ?



Data Set Description for Work.Y2003

Variable	Type	Length
FlightNum	num	8
FirstClass	num	8
BusinessClass	num	8
Coach	num	8

Data Set Description for Work.April

Variable	Type	Length
FlightNum	char	4
FirstClass	num	8
BusinessClass	num	8
Coach	num	8

a. Missing values are assigned to **`Room`** for the observations that are read in from *Work.NewHires*.

b. Missing values are assigned to **`Room`** for all of the observations in *Work.Employees*.

c. **`Room`** is dropped from *Work.Employees*.

d. The values of **`Name`** are truncated in the observations that are read in from *Work.NewHires*.

10. You do *not* need to use the FORCE option with PROC APPEND when ?

a. the DATA= data set contains variables that are not in the BASE= data set.

b. the BASE= data set contains variables that are not in the DATA= data set.

c. the variables in the DATA= data set are longer than the corresponding variables in the BASE= data set.

d. the variables in the DATA= data set have a different type than the corresponding variables in the BASE= data set.

Answers

1. Correct answer: c

When a FILENAME statement is used to assign a fileref to multiple raw data files, the list of files must be enclosed in a single set of parentheses. Each filename specified must be enclosed in quotation marks.

2. Correct answer: d

The FILEVAR= option enables you to dynamically change the currently opened input file to a new input file. The FILEVAR= variable must contain a character string that is a physical filename. Like automatic variables, the FILEVAR= variable is not written to the data set.

3. Correct answer: b

The DO statement creates the index variable **`x`** and assigns it the values of *8, 9*, and *10*. The assignment statement assigns the name of a raw data file to **`readfile`** using the current value of **`x`** and the PUT function, which concatenates the values of **`x`** with the text strings *c:\data\month and .dot*. The COMPRESS function removes blank spaces from the values of **`readfile`**.

4. Correct answer: a

The TODAY function returns the current date from the system clock as a SAS date value. The year number is then extracted from the current date using the YEAR function. The value of the current year, *2003*, is assigned to **`y3`**. The year values *2002* and *2001* are assigned to **`y2`** and **`y1`**, respectively. The PUT function concatenates the text string *c:\data\Ywith* the year values and the text string *.dat*.

5. Correct answer: a

The END= option enables you to name a variable whose value is controlled by SAS. The value of the variable is *0* when you are not reading the last record in an input file and *1* when you are reading the last record in an input file. You can test the value of the END= variable to determine if the DATA step should continue processing. Like automatic variables, the END= variable is not written to the SAS data set.

6. Correct answer: b

PROC APPEND uses the BASE= and DATA= arguments. BASE=*SAS-data-set* names the data set to which you want to add observations. DATA=*SAS-data-set* names the SAS data set containing observations that you want to append to the end of the SAS data set specified in the BASE= argument.

7. Correct answer: d

If a DATA= data set contains variables that are longer than the corresponding variables in the BASE= data set, the FORCE option must be used with PROC APPEND. Using the FORCE option enables you to append the data sets. However, some of the variable values may be truncated in the observations that are read in from the DATA= data set.

8. Correct answer: c

You must use the FORCE option with PROC APPEND when the DATA= data set contains a variable that does not have the same type as the corresponding variable in the BASE= data set.

9. Correct answer: a

PROC APPEND reads only the data in the DATA= SAS data set, not the BASE= SAS data set. When the BASE= data set contains more variables than the DATA= data set, missing values for the additional variables are assigned to the observations that are read in from the DATA= data set.

10. Correct answer: b

The FORCE option does not need to be used if the BASE= data set contains variables that are not in the DATA= data set. The FORCE option must be used if

- the DATA= data set contains variables that are not in the BASE= data set
- the variables in the DATA= data set are longer than the corresponding variables in the BASE= data set
- the variables in the DATA= data set have a different type than the corresponding variables in the BASE= data set.